

# Python. Коллекции

Лекция 2

Преподаватель: Дмитрий Косицин

## Импорт модулей ...

© Dzmitryi Kasitsyn

Fall 2017 • 3

## Встроенные функции ...

© Dzmitryi Kasitsyn

Fall 2017 • 7

## Особенности синтаксиса ...

© Dzmitryi Kasitsyn

Fall 2017 • 10

## Списки и кортежи ...

Списки. Кортежи. Особенности итерирования по нескольким коллекциям.  
Индексация. Слайсы. Функциональная обработка коллекций.

© Dzmitryi Kasitsyn

Fall 2017 • 14

## Словари и множества ...

© Dzmitryi Kasitsyn

Fall 2017 • 28

## Ответы и полезные ссылки ...

© Dzmitryi Kasitsyn

Fall 2017 • 33

# Импорт модулей

...

# Импорт модулей

Импорт модулей:

- `import <module_name> [as <alias>]`
- `from <module_name> import (<name_1> [as <alias_1>],  
...,  
<name_k> [as <alias_k>])`

```
>>> import sys
>>> print(sys.version)
2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC
v.1500 32 bit (Intel)]
```

# Загрузка модулей

Все загруженные модули хранятся в `sys.modules`

```
>>> def is_fractions_imported():  
>>>     return 'fractions' in sys.modules
```

```
>>> print(is_fractions_imported())  
False
```

```
>>> import fractions  
>>> print(is_fractions_imported())  
True
```

Специальный модуль `__future__`

```
>>> from __future__ import division  
>>> print(7 / 3)  
2.333333333333333
```

# Поиск модулей

Очередность мест поиска:

1. `sys.modules` (уже мог быть загружен)
2. `builtins`
3. `sys.path`:
  - current working directory
  - `PYTHONPATH`
  - Python source directory etc.

! Можно модифицировать `sys.modules` и `sys.path`

\* на самом деле чуть сложнее

# Встроенные функции

...

# Встроенные функции

Посмотреть доступные переменные можно вызвав функцию **dir**

```
# просмотр встроенные объектов (функций, исключений и т.п.)  
>>> dir()
```

```
# просмотр атрибутов и методов некоторого объекта  
>>> dir(object)
```



# Встроенные функции

- `bin`, `oct`, `hex` – преобразует число в строку в заданной системе счисления
- `ord`, `chr` – преобразует Unicode-символ (UCS2) в числовой код и обратно
- `bool`, `int`, `str` – стандартные типы (в т.ч. могут использоваться для преобразования типов)
- `abs`, `sum`, `round`, `min`, `max`, `pow`, `divmod` – общие математические функции

```
>>> max(1, 3, 4, 2)  # an iterable may be also passed into  
4
```

Доступные математические модули: `math`, `cmath`, `decimal`, `fractions`, `random` (и `statistics` – Python 3.4+)

# Особенности синтаксиса

...

# Особенности синтаксиса

- В Python есть **break** и **continue**, а также ключевое слово **pass**

```
>>> while True:    # infinite loop  
>>>     pass
```

- Ключевое слово **else** применимо в связке с **if**, так и с **for**, **while** и **try**
- Конструкция **switch** заменяется на **if-elif-...-else**

**Важно!** Наличие **else** крайне рекомендуется. В нем следует либо разместить **pass**, либо бросить исключение.

# Сравнение объектов

Все переменные – указатели на некоторые ячейки памяти.

`id(...)` – возвращает адрес конкретного объекта

Проверка, что `x` и `y` указывают на *один и тот же* объект:

`x is y`

Проверка, `x` и `y` указывают на *равные по значению* объекты:

`x == y`

# Пример сравнения объектов

```
>>> x = [3, 5, 9]
```

```
>>> y = [3, 5, 9]
```

```
>>> print("are objects equal: %s" % (x == y))
```

```
are objects equal: True
```

```
>>> print("are objects same: %s (id of x - %d; id of y - %d)"
```

```
      % (x is y, id(x), id(y)))
```

```
are objects same: False (id of x - <some number>; id of y - <some other number>)
```

```
>>> print("x equals itself comparing values (%s)"
```

```
      " and identifiers (%s)" % (x == x, x is x))
```

```
x equals itself comparing values (True) and identifiers (True)
```

# Списки и кортежи

...

Списки. Кортежи. Особенности итерирования по нескольким коллекциям.

Индексация. Слайсы. Функциональная обработка коллекций.

# Список

Список – изменяемая последовательность объектов

Создание:

```
x = list()      x = []
```

Добавление элементов (изменение списка!):

```
x.append(v)      x.insert(idx, v)  
x += <iterable>  x.extend(iterable)
```

\* *iterable* – некий итерируемый объект, например, список или кортеж

# Индексирование

Взятие элемента:

```
x[idx]
```

Присваивание элемента:

```
x[idx] = v
```

Допускаются отрицательные индексы:

```
>>> x = [100]  
>>> x[0] == x[len(x) - 1] == x[-1]  
True
```

Некорректные индексы порождают исключение **IndexError**.



# Работа со СПИСКОМ

Удаление элементов (изменение списка!):

`x.pop(idx)`      `x.remove(v)`      `del x[idx]`

Изменение порядка элементов:

`x.sort()`      `x.reverse()`

*Вопрос: какая сортировка используется?*

Проверка принадлежности элемента:

`x.index(v)`      `x.count(v)`      `v in x`

Преобразование к **bool**:

`bool([])` вернет **False**

# Работа со списком

Списки можно сравнивать (лексикографически)

```
>>> [1, 2, 3] > [1, 2, 1]  
True
```

Списки можно умножать:

```
>>> [1] * 5 == [1, 1, 1, 1, 1]  
True
```

Получить отсортированную копию списка – **sorted**(x) (built-in функция, поддерживает аргумент *key* для сортировки сложных типов)

Развернуть список – **reversed** (x) (built-in, возвращает *итератор*)

# Кортежи

Кортеж – неизменяемая последовательность объектов

*Отличие от списка:* не допускает присваивания по индексу, добавления, вставки и удаления элементов, а также сортировку и обращение порядка.

Создание:

```
>>> x = (1, 2, 3)    # or just x = 1, 2, 3
```

```
>>> y = tuple([5, 6, 7])
```

```
x += 1, 2    # создает новый кортеж!
```

# Итерирование по нескольким коллекциям

Итерирование по двум коллекциям – функция **zip**:

```
>>> for x, y in zip([1, 2, 3], [-3, -2, -1]):  
>>>     print(x % y == 0)  
False  
True  
True
```

**Важно!** **zip** возвращает новый список кортежей в Python 2.x и генератор в Python 3.x.

**enumerate**(x) – сокращение **zip(range(len(x), x))**

# Слайсы

Можно обращаться сразу к нескольким элементам коллекции:

```
>>> x = list(range(10))
>>> print(x[0:10:2])
0 2 4 6 8
```

**Замечание.** Вспомните `range(0, 10, 2)`

Слайсы за пределами коллекции или некорректные:

```
>>> x[100:110]
[] # тип соответствует типу переменной x
```

```
>>> x[0:10:-2]
[]
```

# Способы задания слайсов

Следующие записи эквивалентны:

- `x[9:0:-2]`
- `x[-1:0:-2]`
- `x[:None:-2]`

Слайс – это объект **slice**

```
>>> x[slice(9, 0, -2)] == x[9:0:-2]
```

**True**

# Замечания по работе со слайсами

Слайсу можно дать имя и связать с переменной

```
>>> even_indices_slice = slice(None, None, 2)
>>> print(x[even_indices_slice])
```

Слайсам можно присваивать

```
>>> x = [1, 2, 3]
>>> x[0:2] = [7, 6, 5]
>>> print(x)
[7, 6, 5, 3]
```

# Замечания по работе со списками

Три способа создать список, содержащий три списка:

```
>>> x = [[], [], []]
>>> y = [[]] * 3
>>> z = []
>>> for _ in range(3):
>>>     z.append([])
([[], [], [], [], [], []])

# использовать ";" (semicolon) нельзя!
>>> x[0].append(1); y[0].append(2); z[0].append(3)
```



# Замечания по работе со списками

```
>>> print(x, y, z)
([[1], [], []], [[2], [2], [2]], [[3], [], []])
```

Wow!.. Лучше использовать другой способ!

```
>>> y_pretty = [[] for _ in range(3)]
>>> y_pretty[0].append(2)
>>> print(y_pretty)
[[2], [], []]
```

Такая конструкция называется list comprehension.

# Функциональный подход

Основные функции для работы с последовательностями:

- **map** – применить функцию к каждому элементу последовательности;
- **filter** – оставить только те элементы, для которых переданная функция возвращает **True** (предикатом может быть **None**);
- **all** – возвращает **True**, если все элементы преобразуются к **True**
- **any** – возвращает **True**, если хотя бы один элемент **True**

**Важно!** В Python 2.x функции **map** и **filter** возвращают списки, в то время как в Python 3.x – генераторы

**Замечание.** Функция *reduce* не рекомендуется к использованию.

# Замечание по comprehension-выражениям

Comprehensions допускают вложенные **for** и одно **if** выражение:

```
>>> def is_odd(x):  
>>>     return bool(x % 2)
```

```
>>> filter(is_odd, range(10)) == [x for x in range(10) if is_odd(x)]  
True
```

Tuple-comprehension нету. Выражение с круглыми скобками – генератор

```
>>> another_comprehension = (is_odd(x) for x in range(10))  
>>> list(another_comprehension) == [is_odd(x) for x in range(10)]  
True
```

# Словари и множества

...

# Словари и хэши

Словарь отражает связь ключ-значение.

В Python словари реализованы как hash-таблица (ассоциативный массив пар).

*Вопрос: какая хэш-функция используется?*

Создание словаря:

```
empty_dict= {}    # or empty_dict= dict()  
x = dict(zip(range(3), range(6, 9)))  
y = {2: 8, 1: 7, 0: 6}
```

# Особенности хэширования

```
>>> hash(1)
1
```

```
>>> hash(True)
1
```

```
# тут все хорошо
>>> hash((1, 2, 3))
>>> hash(tuple())
```

```
#raise TypeError!
>>> hash([1, 2, 3])
>>> hash(x)
```

Важно! Ключи должны быть хэшируемыми (подробнее разберем позднее).

# Работа со словарями

Взятие элемента:

```
x[key]      x.get(key, default)
```

Присваивание элемента (см. также **update**):

```
x[key] = value      x.setdefault(key, default)
```

Удаление элемента(см. также **clear**):

```
del x[key]      x.pop(key, default)
```

Проверка принадлежности:

```
key in x
```

# Особенности словарей

Словари не гарантируют порядок обхода (даже в Python 3.x!)

Словари можно сравнивать на равенство.

Методы для итерирования в Python 2.x:

- **keys()**, **values()**, **items()** – возвращают списки ключей, значений и пар ключ-значение
- **iterkeys()**, **itervalues()**, **iteritems()** – возвращают, соответственно, итераторы
- **viewkeys()**, **viewvalues()**, **viewitems()** – возвращают *view*-объекты

**Важно!** В Python 3.x есть только методы **keys()**, **values()** и **items()**, возвращающие *view*-объекты. Такие объекты отражают изменения в исходной коллекции.



# ОТВЕТЫ И ПОЛЕЗНЫЕ ССЫЛКИ

...

# ОТВЕТЫ

- Сортировка – [TimSort](#) (производная merge sort и insertion sort)
- Hash-функция в Python < 3.4 – FNV, далее – SipHash ([PEP456](#))

# Полезные ссылки

- Сайт Jupyter (инструкция по установке и документация – <https://jupyter.org>)
- Как использовать сразу две версии Python: [здесь](#) и [здесь](#)
- Документация по «магическим» выражениям Jupyter – [здесь](#)